SLAM Demo

EDAA – G06

João Martins
Henrique Ribeiro
João Costa
Tiago Duarte

Simultaneous Location And Mapping

- **Goal** – map an environment navigated by and autonomous vehicle, while simultaneously locating it in the map;
- **Challenges:**
  - Move from 2D space to 3D space - Raycast volume estimation
  - Optimize parallel raycasts - Hash Sets
  - Improve obstacle and edge detection - Canny filter
- **Datasets** – the group will have access to sonar data measured by CRAS.

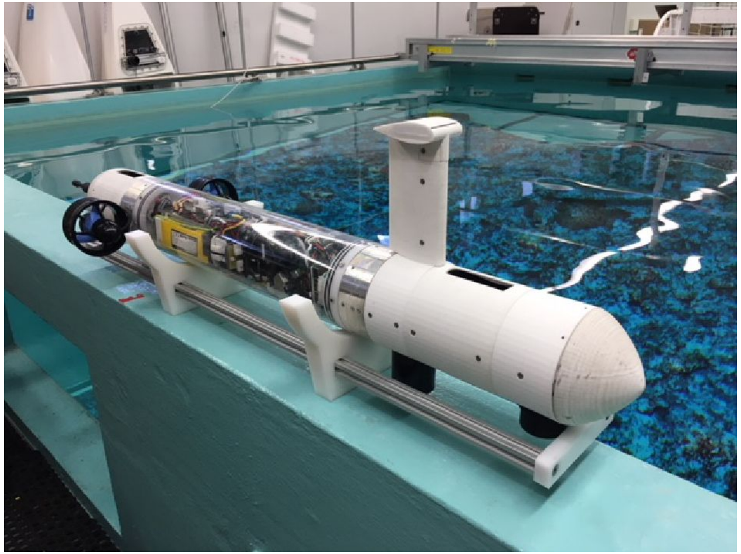
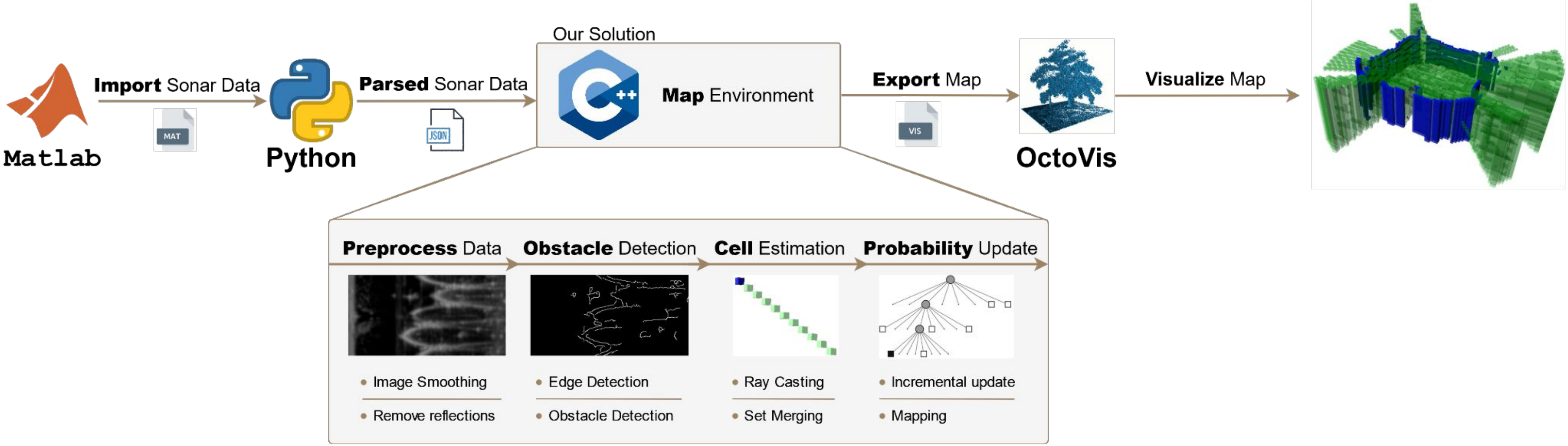


Fig 1. UAV used to collect the datasets.

Pipeline

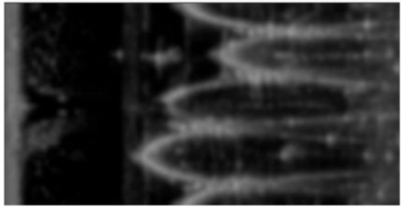
Pipeline



Matlab
Import Sonar Data
MAT
Python
Parsed Sonar Data
JSON
Our Solution
Map Environment
Export Map
VIS
OctoVis
Visualize Map
Preprocess Data
Obstacle Detection
Cell Estimation
Probability Update
Image Smoothing
Remove reflections
Edge Detection
Obstacle Detection
Ray Casting
Set Merging
Incremental update
Mapping

Our Solution

Preprocess Data

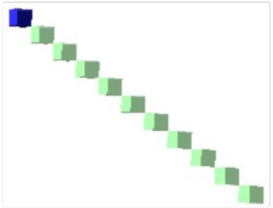


- Image Smoothing
- Remove reflections

Obstacle Detection

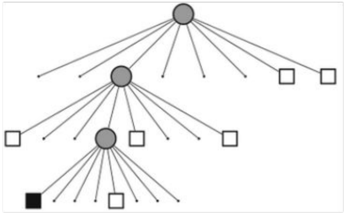
- Edge Detection
- Obstacle Detection

Cell Estimation



- Ray Casting
- Set Merging

Probability Update

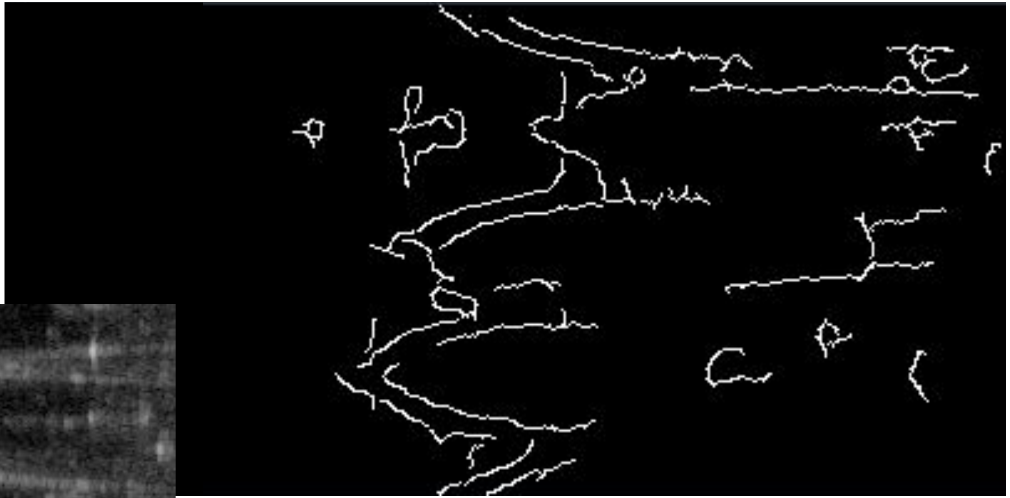


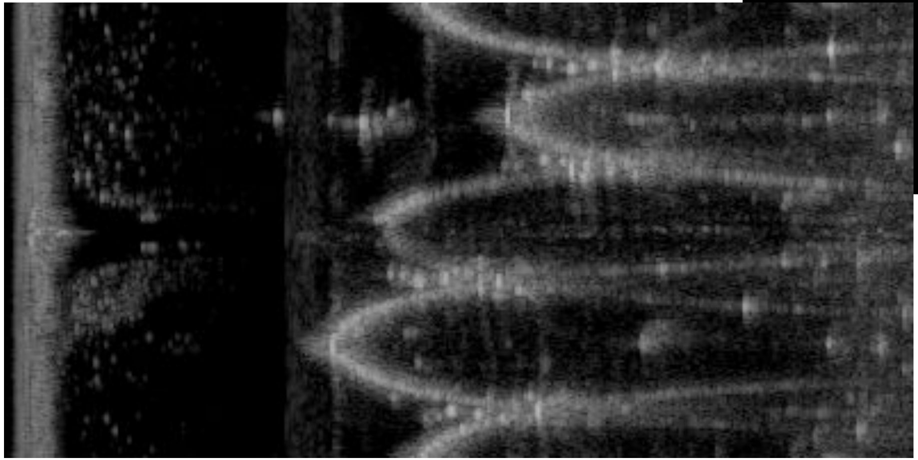
- Incremental update
- Mapping

Obstacle Detection

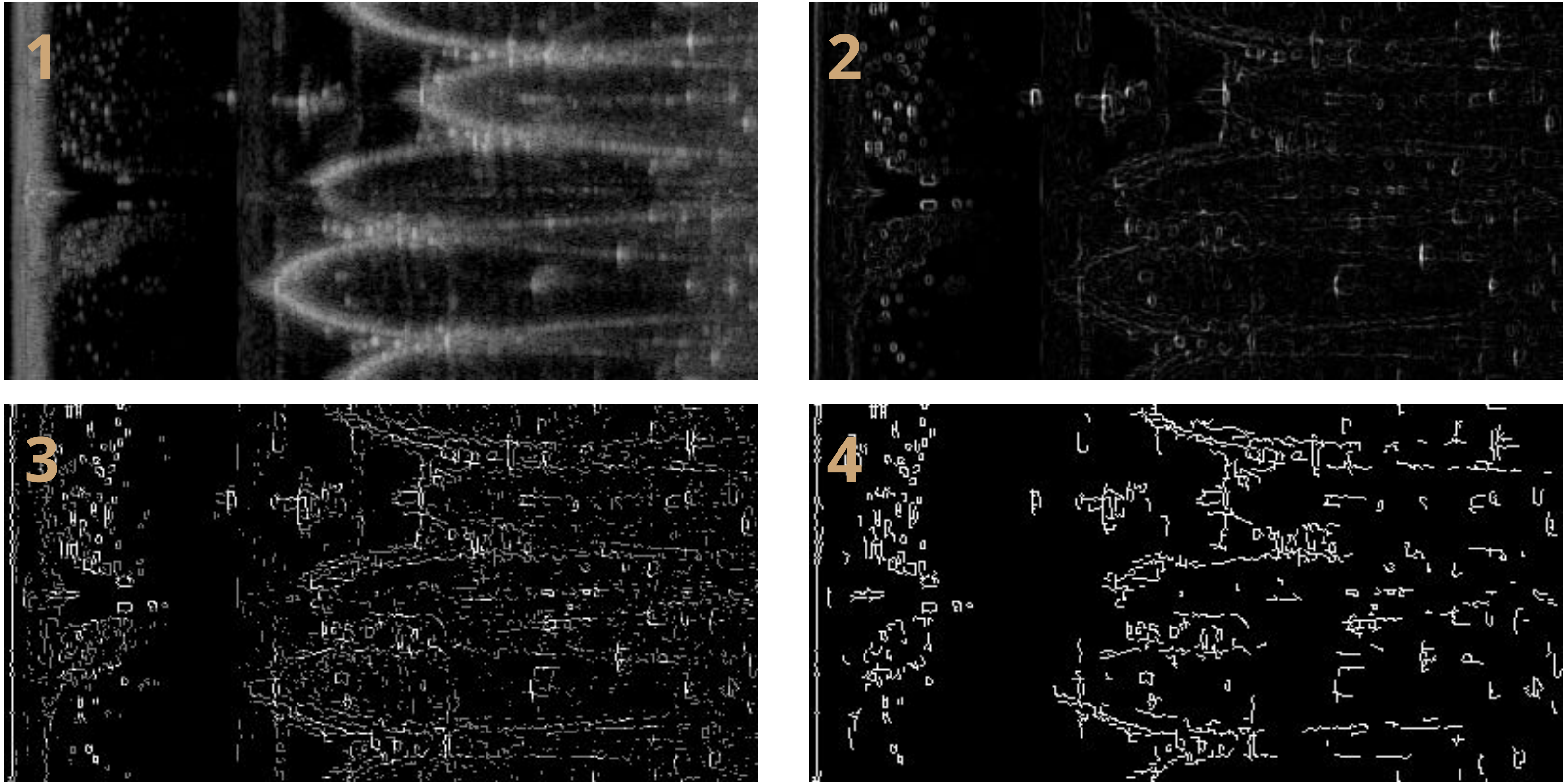
- **Canny Thresholding**

Obstacle Detection - Canny threshold

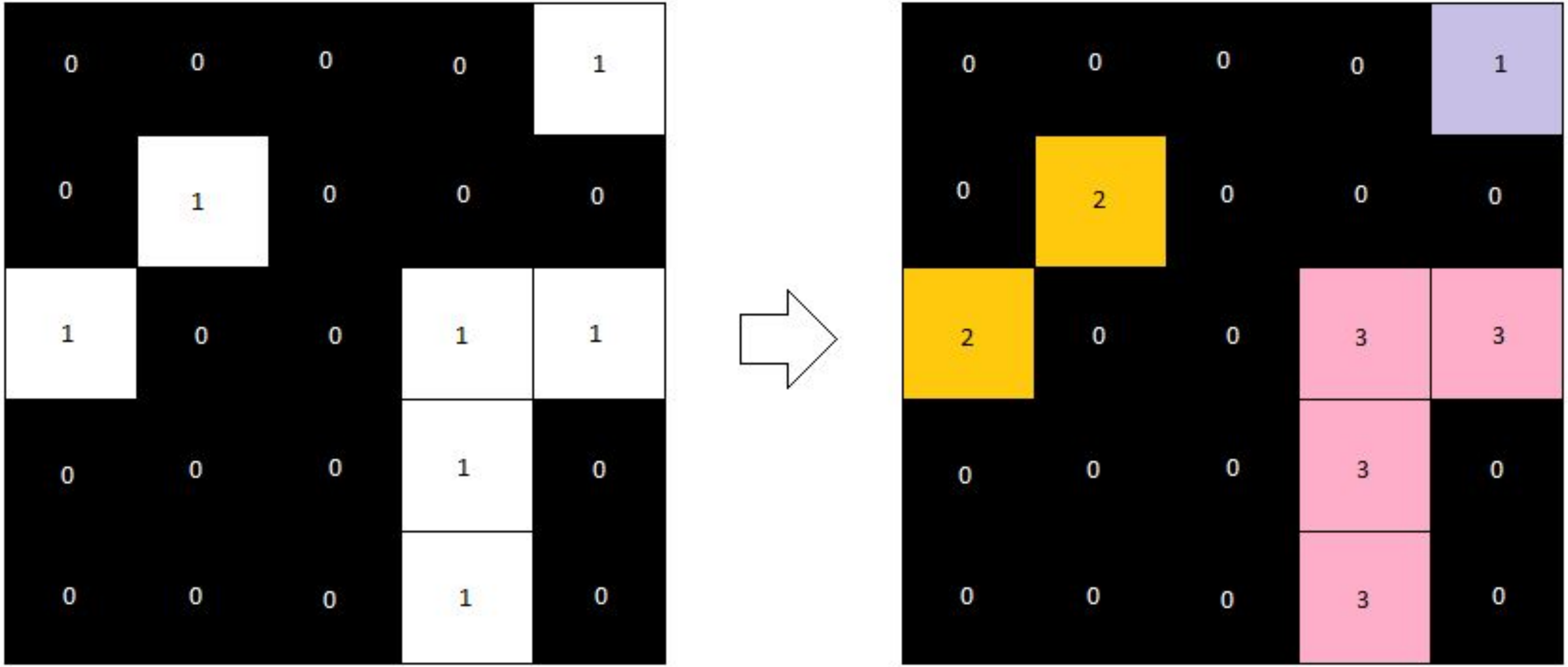




Canny Intermediate Results

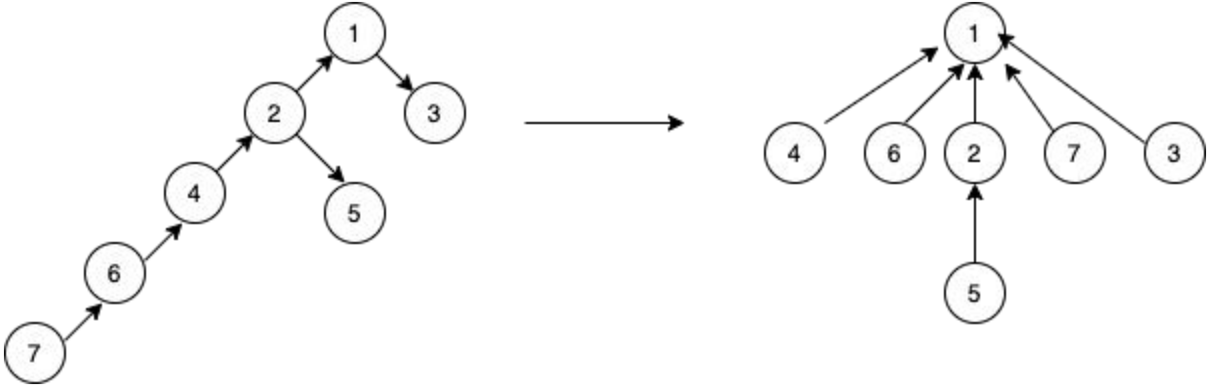


Hysteresis and Blob Analysis



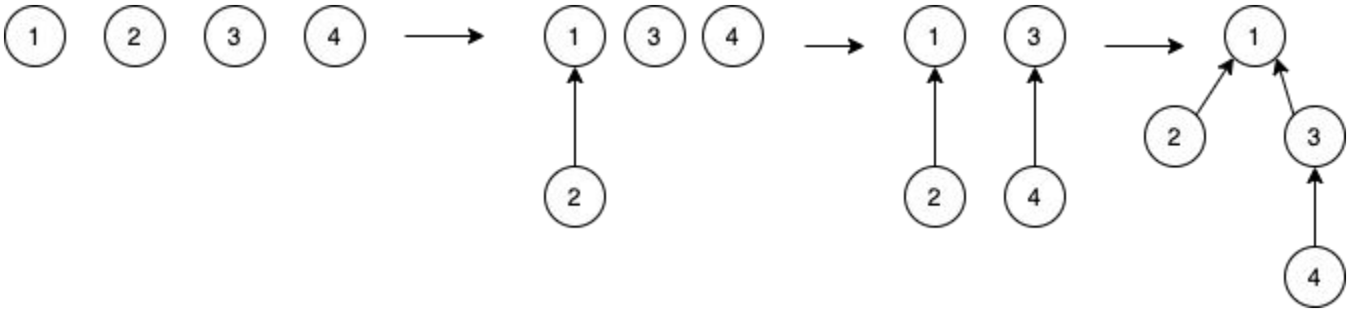
Disjoint Sets

- Elements partitioned into a number of disjoint sets
- Using Path Compression

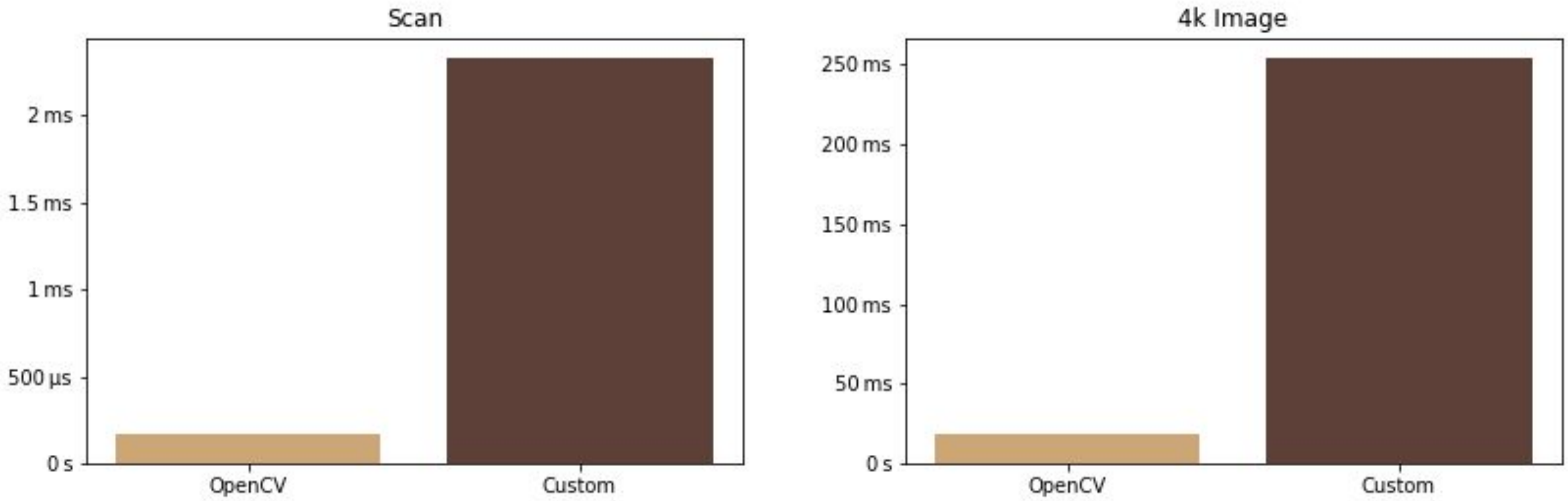


Disjoint Sets - Applied to our Blob Analysis

- Keep the smallest element at the root of each set
- ✖ Cannot implement Union by Rank



Performance



Cell Estimation

- **Ray Casting**
- **Set Merging**

Raycasting

- Bresenham's line algorithm:
    - Travels in all axis at once
    - Handles diagonal transitions very well
    - Very accurate

- Several beams are sent at once
    - Overlapping cells that need to be merged

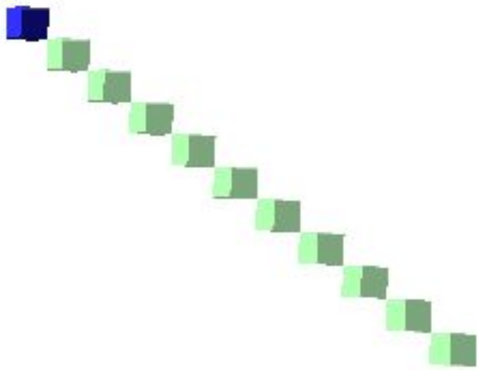


Fig 2. Bresenham's Line Algorithm

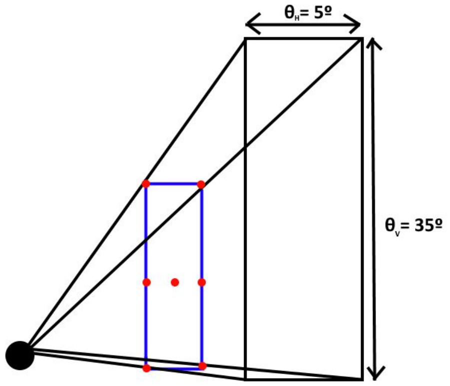


Fig 3. Volume estimation with raycasts

Sets using Hash Maps

- Used to join covered cells from parallel raycasts
- Set merge operation used 60% of ray cast's execution time;

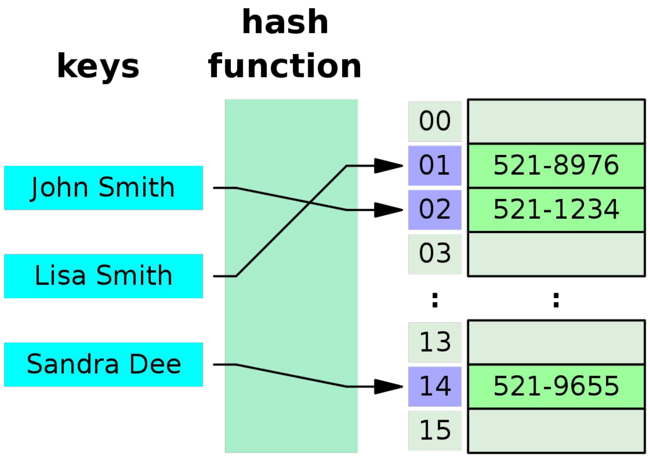


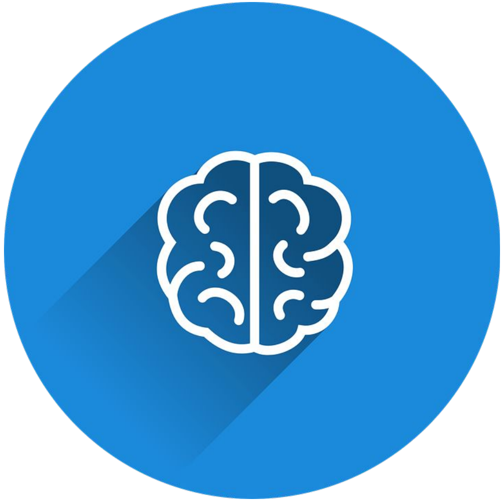
Fig 4. Hash Map Representation

Set/Hash table

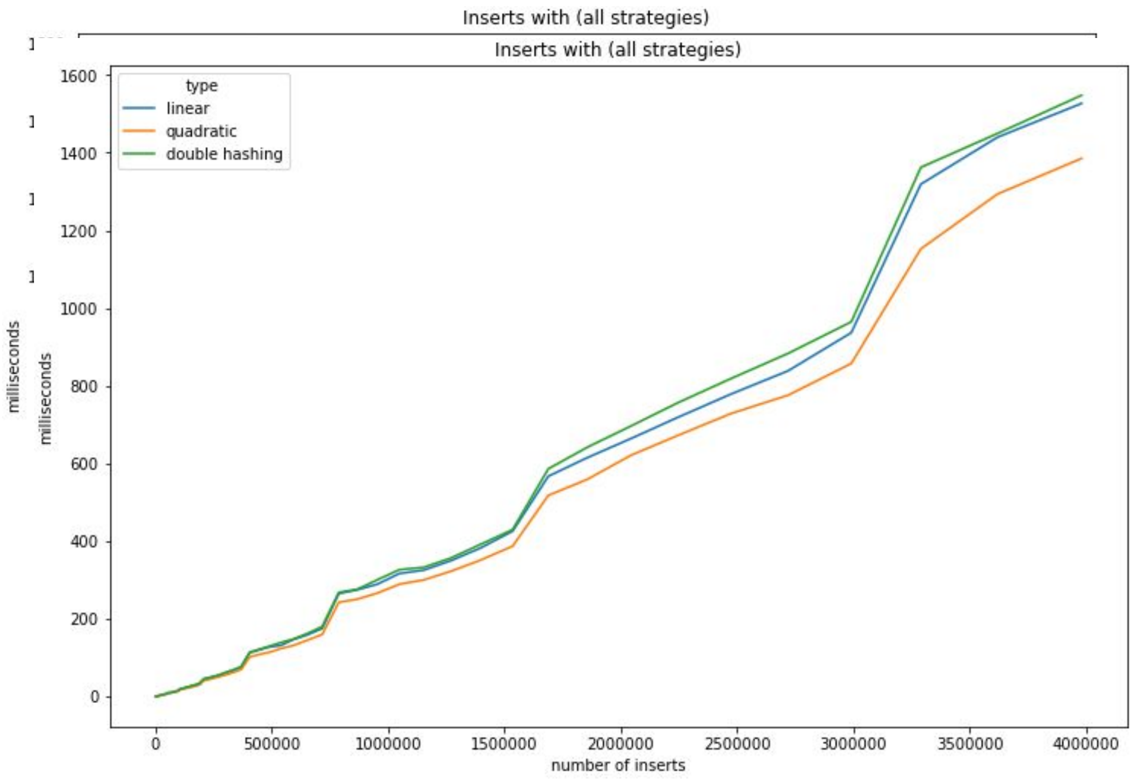
- Implementing sets (based on Hash tables)
- Focus on improving performance on our use case

Rationale – Opportunities for improvements

- Our use case needs unordered sets as containers for very small objects:
  - Objects are 48-bit < 64-bit pointers.
- C++ sets use buckets:
  - Increases memory usage.
- C++ sets aren’t fast for our use case.

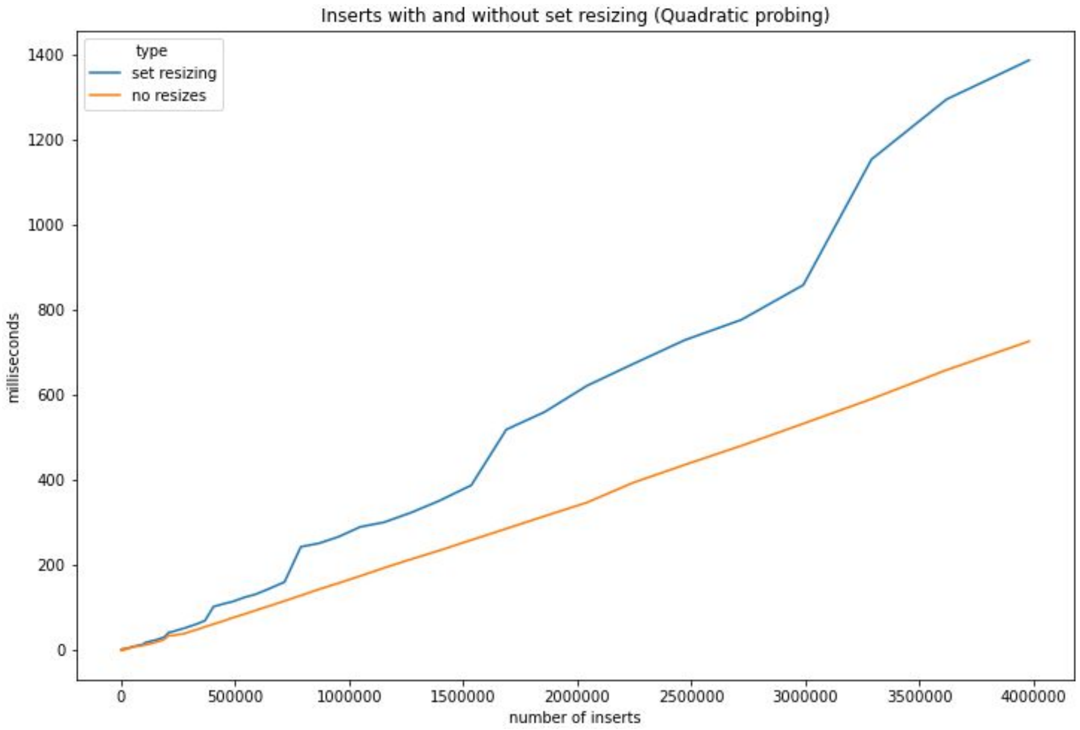


Insert – Comparison of all strategies



- Insertion of randomly generated objects;
- Performance is similar between all methods;
- **Quadratic probing** is consistently the best.

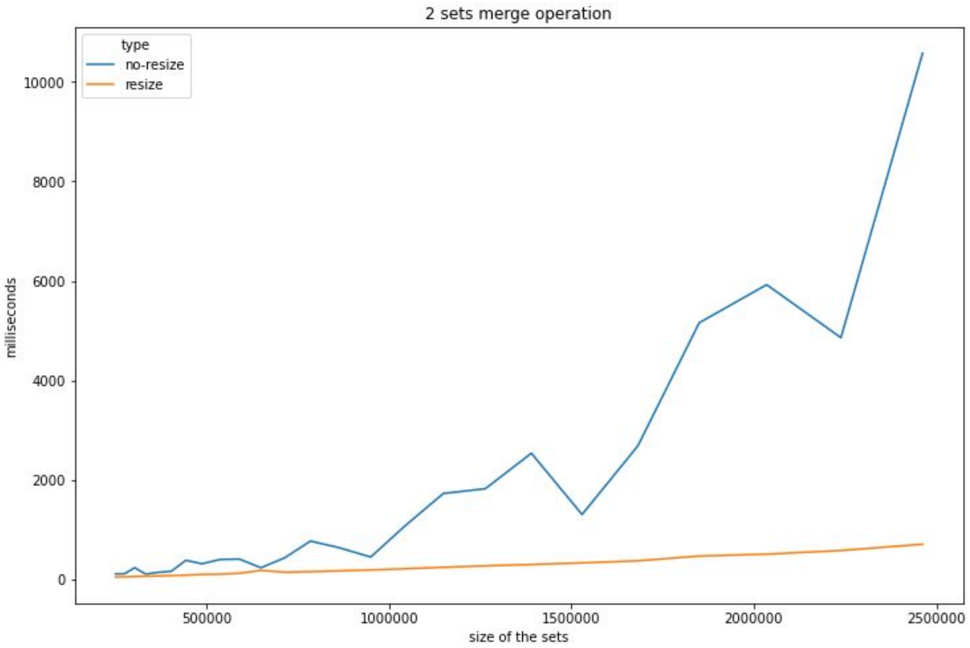
Insert – Impact of set resizing



- In the previous examples, the sets were resized multiple times;
- This also affects the comparisons:
    - Smaller sets suffer fewer resizes.
- Resize is triggered by a threshold of how full the set is (**75%**);

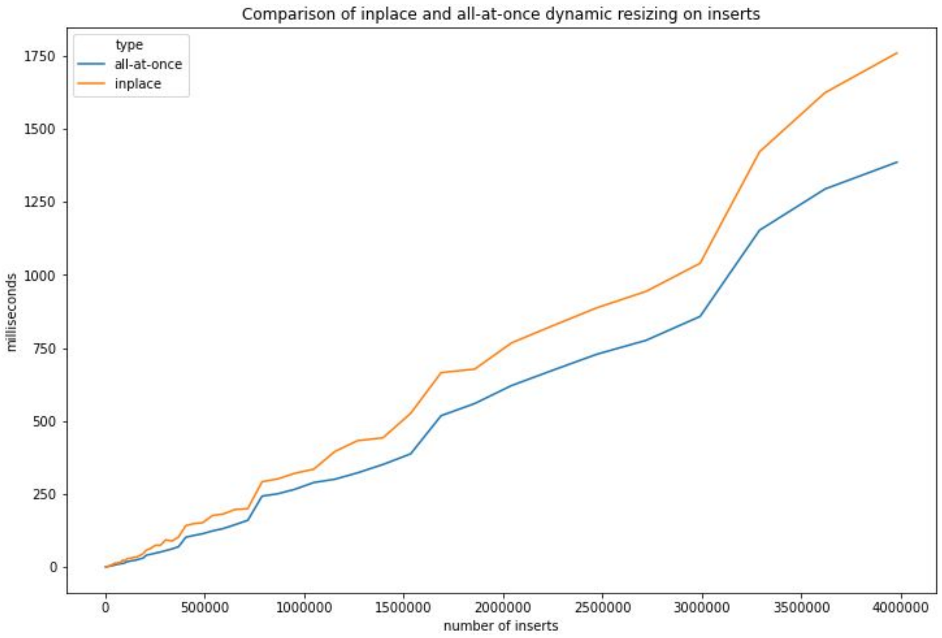
Merge – Performance and impact of resize

- In our project, we only need **in-place merges**;
- The **merge** operation consists of iterating over a set and inserting each object into the second one;
- By doing a **pessimistic assumption**, we can greatly increase the performance of the operation:
  - Assume that each element in the second set is new (not already part of the set);
  - Check that the **load factor is still ok** in that case (no resize needed).



In-place dynamic resizing – comparison

- We implemented a way for sets to be dynamically resized in-place:
  - Less memory overhead;
  - Less time performance.
- Algorithm:
  - Grow the container (double size);
  - Re-evaluate all entries in first half;
  - If they fall on second half, place them there;
  - Otherwise, save them in a buffer for later;
  - Clear first half and insert the buffer elements



Real data test

- Tests on real data:
  - Ray-casts on a plane point cloud dataset;
  - Only considering the turbines;
  - About 1000 ray-casts covering 18000 cells each.
- Ray-casts are calculated 4 at a time in parallel:
  - On a computer with 4 CPU cores.
- Our set implementation **improved performance almost 2-fold**.

	C++ sets	**Our sets**	**In-place resize**
Time taken	≃ 23 s	≃ 13 s	**≃ 10 s**

Probability Update

- Incremental Update

Incremental Probability Update

- Build map probabilities on top of previous sweep measurements

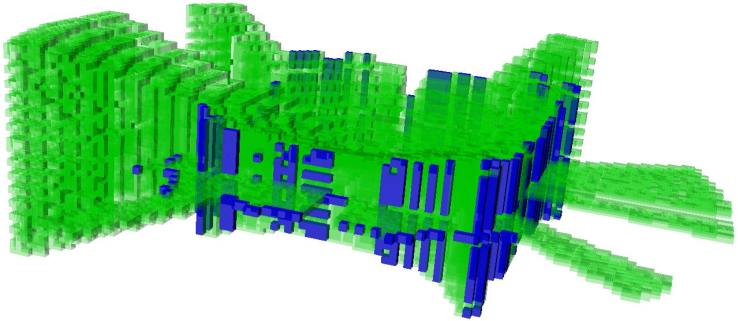


Fig 5. Map after first sweep

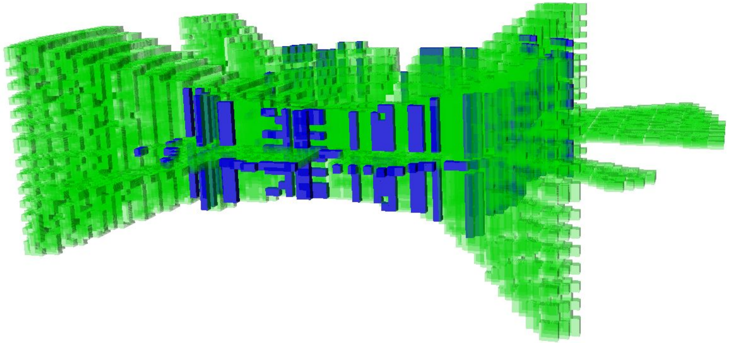


Fig 6. Map after 27 sweeps

Demo

- Results showcasing
- Demo video

Results

Future work

- Aspects to explore
- Things to improve

Future work

- Explore **fast modulo** operation (as defined by Boost);
- Try different set collision resolution techniques;
- Use OpenCV’s Canny algorithm;
- Explore different SLAM datasets;
- Deal with the discontinuities found in the walls of the tank;
- Explore alternatives to raycasts;
- Explore alternative probability distributions.

Slide deck source

[Slide deck]